php

Search

rawurlencode

Change language: French

Edit Report a Bug

(PHP 4, PHP 5)
rawurlencode — Encode une chaîne en URL, selon la RFC 3986

Description

```
string rawurlencode ( string $str )
```

Encode la chaîne fournie, en accord avec la » RFC 3986.

Liste de paramètres

str
L'URL à encoder.

Valeurs de retour

Retourne une chaîne dont tous les caractères non alphanumériques (hormis

```
-_.~
```

) auront été remplacés par des séquences *%xy* (*%*), avec xy, deux valeurs hexadécimales. Ce codage est conforme à la » RFC 3986 qui évite que les caractères spéciaux soient interprétés comme des délimiteurs, et pour protéger les URL lors du transfert (contrairement à certains systèmes email).

Note:
Avant PHP 5.3.0, rawurlencode encodait les tildes (~) suivant la » RFC 1738.

Historique

Version	Description
5.3.4	Les caractères tildes (~) ne sont plus encodés lorsque la fonction **rawurlencode()** est utilisée avec les chaînes EBCDIC.
5.3.0	Conformité avec la » RFC 3986.

Exemples

Exemple #1 Exemple avec rawurlencode()

```
<?php
echo '<a href="ftp://user:', rawurlencode('foo @+%/'),
    '@ftp.example.com/x.txt">';
?>
```

L'exemple ci-dessus va afficher :

```
<a href="ftp://user:foo%20%40%2B%25%2F@ftp.example.com/x.txt">
```

Ou, si vous transmettez un composant PATH_INFO d'une URL :

Exemple #2 Exemple avec rawurlencode()

```
<?php
echo '<a href="http://example.com/department_list_script/',
   rawurlencode('sales and marketing/Miami'), '">';
?>
```

L'exemple ci-dessus va afficher :

```
<a href="http://example.com/department_list_script/sales%20and%20marketing%2FMiami">
```

Voir aussi

- rawurldecode() - Décode une chaîne URL
- urldecode() - Décode une chaîne encodée URL
- urlencode() - Encode une chaîne en URL
- » RFC 3986

User Contributed Notes 22 notes

add a note

▲ 11 ▼ bolvaritamas at vipmail dot hu 3 years ago

I've written a simple function to convert an UTF-8 string to URL encoded string. All the given characters are converted!

The function:

```
<?php
function mb_rawurlencode($url){
$encoded='';
$length=mb_strlen($url);
for($i=0;$i<$length;$i++){
$encoded.='%'.wordwrap(bin2hex(mb_substr($url,$i,1)),2,'%',true);
}
return $encoded;
}
?>
```

Example:

```
<?php
echo 'http://example.com/',
   mb_rawurlencode('你好');
?>
```

The above example will output:
http://example.com/%e4%bd%a0%e5%a5%bd

▲ 1 ▼ hiroaki dot kawai at gmail dot com 5 years ago

phpversion()>=5.3 will compliant with RFC 3986, while phpversion()<=5.2.7RC1 is not compliant with RFC 3986.

History of related RFCs:

RFC 1738 section 2.2
only alphanumerics, the special characters "$-_.+!*'(),", and
reserved characters used for their reserved purposes may be used
unencoded within a URL.

RFC 2396 section 2.3
unreserved = alphanum | mark
mark = "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"

RFC 2732 section 3
(3) Add "[" and "]" to the set of 'reserved' characters:

RFC 3986 section 2.3
unreserved = ALPHA / DIGIT / "-" / "." / "_" / "~"

RFC 3987 section 2.2
unreserved = ALPHA / DIGIT / "-" / "." / "_" / "~"

▲ 2 ▼ piotr DOT wydrych AT gmail DOT com 8 years ago

You can encode paths using:

```
<?php
$encoded = implode("/", array_map("rawurlencode", explode("/", $path)));
?>
```

▲ 2 ▼ Anonymous 11 years ago

Note that RFC 1738 has been amended:
The "[" and "]" are no longer considered unsafe, but instead are now considered "reserved", meaning that they CAN be used in URLs!

Currently this usage has only been allowed in the hostname part, but there are some proposals to allow such use in some URL schemes. Similar extensions are now found that use the "{}" characters as "reserved" characters with special semantics, instead of "unsafe" characters that must be URL encoded...

Note also that some characters are currently "reserved" but should have instead been considered as "unsafe": this includes the parenthesis "()" which are clearly unsafe when a URL is used in MIME headers.

Because of this, if a valid URL contains "()" characters, one should use an upper-level encoding to either enclose the URL with a pair of "unsafe" characters defined in the upper-level protocol (for example a "<>" pair in MIME headers, because these characters cannot be part of a valid URL)...

▲ 1 ▼ Anonymous 11 years ago

--- 1) About "reserved" characters in URLS:

Beware that RFC 1738 specifies that the characters "{", "}", "|", "\", "^", "~", "[", "]", and "`" are all considered unsafe and SHOULD be URL-encoded with a "%xx" triplet within *ALL* URLs.

However, some HTTP URLs seem to use the "~" character as a prefix for a user account for example:
http://www.any.host.domain/~user/subpath/page.html?query#fragment

This usage is acceptable, but the RFC specifies that "%7E" should be used instead of "~" in the path component. HTTP servers should accept "~" as being equivalent to "%7E", and according to the RFC, the "%7E" form should be the canonical one.

However, some HTTP servers are not fully complying to this RFC and consider "%7E" differently from "~" (i.e. they consider it as being part of a path component name, and search a directory name containing a "~" character, instead of mapping the "~user" path component to a user's directory. In that case, these non compliant HTTP server will not find the resource associated to that URL and may return a 404 error or other errors such as an access denied.

When using rawurlencode() on such HTTP URLs, it's best to consider this legacy usage, by using str_replace() on the result to convert back "/%7E" to "/~", so that the URLs will correctly map to the legacy use of the "~" character by these servers. On compliant HTTP servers, they will treat the "~" unsafe character equivalently with the "%7E" recommanded form, so they will automatically canonicalize the "~" character into "%7E".

--- 2) Encoding of hostnames in URLs

Finally, beware that host domain names parts in URLs *MUST NOT* be encoded with rawurlencode(), as the "[" and "]" are valid delimiters that *MUST* be used to reference an IPv6 address or other hostnames that don't fit to the restricted set of characters allowed in a host name (the "[" and "]" characters MUST be used if the hostname includes characters such as ":" which is typically used to specify an alternate non-default port number).

The encoding of host names uses another encoding, required to encode international domain names, with a base-64 encoding of Unicode characters and a "bq--" prefix. This encoding must be used only on individual subdomain parts (separated by "." characters). This encoding does not use any "%xx" triplets.

So NEVER use urlencode() or rawurlencode() on an unparsed URL, unless this full URL is part of a query parameter string!

--- 3) Encoding of username/passwords in URLs:

There is no standard to specify a password in a URL. In fact, there's a legacy usage of the ":" character to separate a username from a password, but it is strongly discouraged. The RFC does not attempt to specify a semantic to the authentication part of an URL (before the "@" character and the hostname part).

If you need to encode a password, always use rawurlencode() on username and passwords separately, and then insert the ":" character to separate both components. Don't use urlencode() (which could use a "+" to encode a space, and would not work because usernames and passwords consider "+" and spaces as being different!)

▲ 0 ▼ Anonymous 2 years ago

If you, like me, sometimes have the misfortune of being forced to work with PHP4, here is a PHP implementation of http_build_query() that produces more or less the same output as this function, accepting the same arguments.

The only differences here are that the RFC selector argument does not behave precisely correctly. This implementation passes RFC1738 through urlencode() and RFC3986 through rawurlencode(), which is not 100% correct, see the manual pages of those function for more information.

```
<?php

 if (!function_exists('http_build_query')) {

   if (!defined('PHP_QUERY_RFC1738')) define('PHP_QUERY_RFC1738', 1);
```

```
    if (!defined('PHP_QUERY_RFC3986')) define('PHP_QUERY_RFC3986', 2);

    function http_build_query ($query_data, $numeric_prefix = NULL, $arg_separator = NULL, $enc_type = PHP_QUERY_RFC1738, $base
= NULL) {
      $result = array();
      $arg_separator = ($arg_separator != '') ? (string) $arg_separator : ini_get('arg_separator.output');
      $enc_func = ($enc_type == PHP_QUERY_RFC3986) ? 'rawurlencode' : 'urlencode';
      foreach ($query_data as $key => $item) $result[] = (is_array($item) || is_object($item)) ? http_build_query($item, NULL,
$arg_separator, $enc_type, ($base !== NULL) ? "$base%5B".$enc_func($key).'%5D' : $enc_func($key)) : (($base !== NULL) ?
"$base%5B".$enc_func($key).'%5D='.$enc_func($item) : ((is_int($key) && $numeric_prefix !== NULL) ? (string) $numeric_prefix :
'').$enc_func($key).'='.$enc_func($item));
      return implode($arg_separator, $result);
    }

  }
```

0 herenvardo at gmail dot com 5 years ago

I have to mention something about javier's post: the issue you are experiencing only happens because you are using ISO-8859-1 (a.k.a. ISO-LATIN-1) encoding, which is an extension of ASCII using the values 128-255 for latin specific characters (these characters are NOT part of ASCII). To say somethig like 0xF1 is the correct value for "ñ" in ASCII is wrong: any value equal or higher than 0x80 is invalid in ASCII; and there is no "correct" value for "ñ" in ASCII because the ASCII character set does not include that character.
These encode/decode functions are designed to work on UTF-8, which is an ASCII-compatible encoding for Unicode, thus being able to represent the entire Unicode character range.

The main point is: the "Ã±" you get is the 0xC3 0xB1 sequence, interpreted as two single-byte ISO-8859-1 characters; but if you interpret them as UTF-8, they indeed represent "ñ". If you are working with the latin character set and encoding, then you are fine with your method (which is essentially a utf-8 => iso-latin-1 converter).

For anybody who is using UTF-8 enconding, check if there is any issue before you use a method like javier's: these multi-byte values are actually the right way to represent any non-ASCII character on UTF-8.

For deeper details on the UTF-8 and ISO-8859-1 encodings, take a look at wikipedia:
http://en.wikipedia.org/wiki/UTF-8
http://en.wikipedia.org/wiki/ISO-8859-1

0 javier dot alejandro dot segura at gmail dot com 6 years ago

```
<?php
/*
 :: Latin characters issue using rawurldecode() ::
   ----------------------------------------

What happen if you need tu convert this %C3%B1 into this 'ñ' using rawurldecode()? Well, it doesn't work as we'd wish to. We'll get this
"Ã±". To fix this issue, I've made the following function:

*/

   function urlRawDecode($raw_url_encoded)
   {
      # Hex conversion table
      $hex_table = array(
         0 => 0x00,
         1 => 0x01,
         2 => 0x02,
         3 => 0x03,
         4 => 0x04,
         5 => 0x05,
```

```
        6 => 0x06,
        7 => 0x07,
        8 => 0x08,
        9 => 0x09,
        "A"=> 0x0a,
        "B"=> 0x0b,
        "C"=> 0x0c,
        "D"=> 0x0d,
        "E"=> 0x0e,
        "F"=> 0x0f
    );

    # Looking for latin characters with a pattern like this %C3%[A-Z0-9]{2} ie. -> %C3%B1 = 'ñ'
    if(preg_match_all("/\%C3\%([A-Z0-9]{2})/i",$raw_url_encoded,$res))
    {
        $res = array_unique($res = $res[1]);
        $arr_unicoded = array();
        foreach($res as $key => $value){
            $arr_unicoded[] = chr(
                (0xc0 | ($hex_table[substr($value,0,1)]<<4)) | (0x03 & $hex_table[substr($value,1,1)])
            );
            $res[$key] = "%C3%" . $value;
        }

        $raw_url_encoded = str_replace($res,$arr_unicoded,$raw_url_encoded);
    }

    # Return raw url decoded
    return rawurldecode($raw_url_encoded);
  }

# Testing
print "Decoded character -> " . urlRawDecode("%C3%B1");

// ouput:
//        Decoded chracter -> ñ

/*

:: A little explanation about what does this function do ::
  ----------------------------------------------------------

This function makes two binary operations between C3 and B1. To get an ASCII representation of this kind of raw url encoded character, we have to make a logical OR between HIGH nibble of 0xC3 (0xC) and HIGH nibble of 0xB1 (0xB) -> (0xC0 | 0xB0), then, a logical AND between both LOW nibble (0x03 & 0x01), and finally we have to make a logical OR between these two results -> [hex] ((0xC0 | 0xB0) | (0x03 & 0x01)) = [binary] ((1100 0000 | 1011 0000) | (0000 0011 & 0000 0001)) = [hex] 0xF1 = [binary] 1111 0001 = "ñ" character.

Hope to be helpfull, if you have an issue like this, try to use this function.

Bye,

Javi =)

*/
?>
```

▲ 0 ▼ christian _no interest in at spam dot com 7 years ago

I had serious trouble with local Windows paths containing umlauts on my Apache 2 / Windows NT machine. Apache could not find any of those files if I just used rawurlencode. It's not noted anywhere here, but you fix this by simply making your path utf8 first:

rawurlencode(utf8_encode($str));

0 Tomek Perlak [tomekperlak at tlen pl] 7 years ago

Note in regards to 'rickyale at ig dot com dot br' program:

Wouldn't the whole issue be fixed by using charset=utf-8 in the HTML page?

I'm passing some data between the HTML form and an PHP program - my 'special' characters have to do with the Polish alphabet - and it looks like JavaScript encoding actually... works.

Of course, I could have tested only a limitted number of cases.

Just a thought.

0 jeroen at virtualhost dot nl 8 years ago

On the comments of rickyale and djmaze...
Is what you try to achieve is not a combination of utf8 and url encoding, e.g. :

```
<?
$str = "bl&#65533;f Charl&#65533;ne";
$enc = urlencode(utf8_encode($str));
$str2 = utf8_decode(urldecode($enc));
echo "$str -> $enc -> $str2";
?>
```

will output:
bl&#65533;f Charl&#65533;ne -> bl%C3%B8f+Charl%C3%A8ne -> bl&#65533;f Charl&#65533;ne

At least works for me, Jeroen Hofstee

0 djmaze@dragonflycms 8 years ago

Easier version to 'rickyale at ig dot com dot br' his example

```php
<?php
function encode($text)
{
   $REQUEST_URI = str_replace('"', '%22', $text);
   // 0 - 128
   return preg_replace('#([\x3C\x3E])#e', '"%".bin2hex(\'\\1\')', $text);
}
?>
```

Just fill the regular expression with all characters that you need to have encoded.

NOTE: 142 and up in his array are language specific ASCII characters so the conversion to their unicode ('%C5%BD') equivelant may or may not work for you.
This needs a far more serious and bigger system to handle for non us tables

0 rickyale at ig dot com dot br 8 years ago

As peter@nospam said, the microsoft uses an different table for encode string when sending data...

with some test i have created a table with this encodes for special char like &#65533; &#65533; &#65533; &#65533; &#65533;

here is it for those who need know what is this table..

the index of array is the ord() of a character..
use with chr(index) to know the char.. and replace with the value.....

var $ENCODE_TABLE = ARRAY(33=>'%21', 35=>'%23', 36=>'%24', 37=>'%25', 38=>'%26', 40=>'%28', 41=>'%29', 43=>'%2B', 44=>'%2C', 47=>'%2F', 58=>'%3A', 59=>'%3B', 60=>'%3C', 61=>'%3D', 62=>'%3E', 63=>'%3F', 91=>'%5B', 92=>'%5C', 93=>'%5D', 123=>'%7B', 124=>'%7C', 125=>'%7D', 142=>'%C5%BD', 192=>'%C3%80', 193=>'%C3%81', 194=>'%C3%82', 195=>'%C3%83', 196=>'%C3%84', 197=>'%C3%85', 199=>'%C3%87', 200=>'%C3%88', 201=>'%C3%89', 202=>'%C3%8A', 203=>'%C3%8B', 204=>'%C3%8C', 205=>'%C3%8D', 206=>'%C3%8E', 207=>'%C3%8F', 210=>'%C3%92', 211=>'%C3%93', 212=>'%C3%94', 213=>'%C3%95', 214=>'%C3%96', 217=>'%C3%99', 218=>'%C3%9A', 219=>'%C3%9B', 220=>'%C3%9C', 221=>'%C3%9D', 224=>'%C3%A0', 225=>'%C3%A1', 226=>'%C3%A2', 227=>'%C3%A3', 228=>'%C3%A4', 229=>'%C3%A5', 231=>'%C3%A7', 232=>'%C3%A8', 233=>'%C3%A9', 234=>'%C3%AA', 235=>'%C3%AB', 236=>'%C3%AC', 237=>'%C3%AD', 238=>'%C3%AE', 239=>'%C3%AF', 242=>'%C3%B2', 243=>'%C3%B3', 244=>'%C3%B4', 245=>'%C3%B5', 246=>'%C3%B6', 249=>'%C3%B9', 250=>'%C3%BA', 251=>'%C3%BB', 252=>'%C3%BC', 253=>'%C3%BD', 255=>'%C3%BF');

example::

```
function encode($text) {
   while(list($ord, $enc) = each($ENCODE_TABLE)) {
   $text = str_replace(chr($ord), $enc, $text);
   }
return $text;
}
```

hope this help ...

▲ 0 ▼ ulterior AT one.lt 8 years ago

In addition to my last post I would like to add that, this function is for the "directories/somefile.ext" paths

In order to construct valid ftp url (with password added in it)
do this

$valid_path = "ftp://" . rawurlencode($user) . ":" . rawurlencode($pass) . ftp_url_encode($your_server_path_to_file)

Last function will encode path url so that language characters remain untouched and you get same file name for download after download dialog appears.

▲ 0 ▼ Anonymous 11 years ago

rawurlencode() MUST not be used on unparsed URLs.

rawurlencode() should not be used on host and domain name parts (that may include international characters encoded in each domain part with a "q--" prefix followed by a special encoding of the international domain, currently in testbed).

rawurlencode() may be used on usernames and passwords separately (so that it won't encode the ':' and '@' separators).

rawurlencode() must not be used on paths (that may contain '/' separators): the ['path'] element of a parsed URL must first be exploded into individual "directory" names. A directory or filename that contains a space must not be encoded with urlencode() but with this rawurlencode(), so that it will appear as a '%20' hex sequence (not '+')

rawurlencode() must not be used to encode the ['query'] element of a parsed URL. Instead you must use the urlencode() function:

Typical queries often use the '&' separator between each parameter. This '&' separator however is just a convention, used in the www-url-encoded format for HTML forms using the default GET method. However, when references are done in a HTML page to an URL that contains static query parameters, these '&' separators should be encoded in the HTML code as '&' for HTML conformance. This is not part of the

URL specification, but of the HTML encapsulation! Some browsers forget this, and send '&' with their HTTP GET query. You may wish to substitute '&' by '&' when parsing and validating URLs. This should be done BEFORE calling urlencode() on query parts.

The ['fragment'] part of a parsed URL (after the first '#' separator found in any URL) must not be encoded with this rawurlencode() function but instead by urlencode().

Validating a URL sent in a HTTP request is then more complicated than what you may think. This must be done only on parsed URLs (where the basic elements of an URL have been splitted), and then you must explode the path components, and check the presence of '&' sequences in the query or fragment parts.

The next thing to do is to check the URL scheme that you want to support (for example, only 'http', 'https', or 'ftp').

You may wich to check the ['port'] part to see if it's really a decimal integer between 1 and 65535.
You may wish to remove the default port number used by the URL schemes you want to support (for example the port '80' for 'http', the port '21' for 'ftp', the port '443' for 'https'), and restrict severely all port numbers below 1024, or some critical ports below 140 (this includes DNS and NetBios ports).

Then you may wish to control severely the ['host'] part (in fact a full host domain name or an IP address), by forbidding those host names that don't contain at least one dot, forbidding those that start with a dot, those that contain two consecutive dots, those that start or finish with a '-' dash, those that contain '.-' or '-.' (invalid in all domain names), those that contain two dashes in another position than the second and third character of a domain name part and not folled by at least one other character, forbid top level domain names that have only one non numeric character, or more than 6 characters (".museum" is, for now, the longest acceptable TLD), check that pseudo-TLD names that are pure integers are effectively between 0 and 255, in that case check that this is a valid IPv4 address by comparing it to long2ip(ip2long($host)), ...

This done, you must use the urlencode() function on all parts up to the exploded path elements, and rawurlencode() on the query and fragment parts, according to the specs, to recreate a complete and validated URL.

▲ 0 ▼ dphantom at ticino dot com 12 years ago

PHP's functions rawurlencode() and urlencode(), both encode the whole argument parameter string, making the result useless as a valid link.

The function listed here encodes a link string (e.g. http://www.domain.com/long_path/to\file.php?query=param#fragm) to a valid <a href=""> parameter string, preserving the original URI structure and the path given.

```
function linkencode ($p_url) {
   $ta = parse_url($p_url);
   if (!empty($ta[scheme])) { $ta[scheme].='://'; }
   if (!empty($ta[pass]) and !empty($ta[user])) {
         $ta[user].=':';
         $ta[pass]=rawurlencode($ta[pass]).'@';
   } elseif (!empty($ta[user])) {
      $ta[user].='@';
   }
   if (!empty($ta[port]) and !empty($ta[host])) {
      $ta[host]=''.$ta[host].':';
   } elseif    (!empty($ta[host])) {
      $ta[host]=$ta[host];
   }
   if (!empty($ta[path])) {
      $tu='';
      $tok=strtok($ta[path], "\\/");
      while (strlen($tok)) {
         $tu.=rawurlencode($tok).'/';
         $tok=strtok("\\/");
      }
      $ta[path]='/'.trim($tu, '/');
```

```
    }
    if (!empty($ta[query])) { $ta[query]='?'.$ta[query]; }
    if (!empty($ta[fragment])) { $ta[fragment]='#'.$ta[fragment]; }
    return implode('', array($ta[scheme], $ta[user], $ta[pass], $ta[host], $ta[port], $ta[path], $ta[query], $ta[fragment]));
}
```

-1 peter at nospam dot webgains dot com 8 years ago

The microsoft URLEncode method ignores the documentation in RFC1738 which states that:

".... the special characters "$-_.+!*'(),", and reserved characters used for their reserved purposes may be used unencoded within a URL"

So for example, myaddress@mydomain.com becomes myaddress%40mydomain%2Ecom, whereas php and other languages would encode this as myaddress%40mydomain.com

This can be an issue when porting from asp or if you are doing string comparison of strings urlencoded on different platforms.

NB. php will correctly decode myaddress%40mydomain%2Ecom to myaddress@mydomain.com, it is only the encoding that differs

-1 Anonymous 11 years ago

About the ";" reserved character in URLs:

rawurlencode() will encode it with a "%2A" triplet. When used on the path part of a URL, this will break the usage defined in URL RFCs, that allows specifying additional parameters to *EACH* element of a path (separated by "/").

So if a path element contains a ";" character (some filesystems allow it, but this is not recommanded) as part of a directory name, this character must be encoded so that it won't be mixed with a parameter extension.

This mapping is allowed on URLs that use a hierarchical scheme (HTTP, HTTPS, FTP, FILE, ...), so that each path element prefixed by "/" can have additional navigational parameters such as authorization strings or semantic parameters.

The generic format of a path element may include path elements such as:
"/." or "/.." or "/.specialname" or "/regularname"
Each part may be followed by a ";" and other parameters separated by ";". These parameters can be eithger ordered or unordered.
Unordered parameters have a symbolic name separated from their value with an equal sign.

Do not mix path element parameters with a query string: these parameters are directly attached to the individual path element, and this makes a difference when this path element is not the last one of the URL. These parameters are part of the resource name (unlike the query string), and the semantic of "." and ".." apply to the full path element with its parameters, so that:
"/subdir1/subdir2/page.html;charset=UTF-8/../index.html"
will resolve to "/subdir1/index.html".

Note that:
"/subdir1/subdir2/page.html;charset=UTF-8"
designates a DISTINCT resource name from:
"/subdir1/subdir2/page.html"
It does not necessarily involves a query, and so it can be cached by default (unlike URLs that contain a query string).

When using path element parameters, their optional name and required value must be rawurlencode()'d separately before inserting ";" and "=" parameters and creating the path elements that will be imploded in the full path.

The consequence is that you MUST not urlencode() or rawurlencode individual path elements, without first parsing them:
- first explode the path into its path lements separated by "/"
- then explode each path element in their name and parameters separated by ";" characters
- then split path element parameters that contain a "=" sign into a name/value pair.
- make sure that unordered path paremeters (that have been cut according to "=" into a pair) are specified *after* ordered parameters (including the main path element name) in each path element, and that no two unordered parameters have the same name (this restriction

does not occur on unordered, unnamed parameters which only supply a value).
- finally you can interpret rawurlencoded names and values that constitute each path element.

Note also that some non-compliant HTTP servers consider that named parameters are ordered, and don't add a semantic to the ";" and "=" used to break up the list of path element parameters. On client agents, when validating URLs, it's best then not to try to interpret this list, and you should just split the main part of a path element and the parameters list by isolating the first ";" that introduces this list. However, the encoded parameter list cannot include any "/" parameter.

Caveats: note that path element parameters (introduced by ";") may be used on upper levels of a hierarchic URL, even before the final document name and its query parameters. When building lists of URLs, you should not separate URLs blindly with a ";" separator, as each URL may include a ";" character, in their path part (the ";" character cannot ocur safely in a query string). In that case, use a surrounding pair such as "<>" or quotes to enclose each URL in such a list.

▲ -2 ▼ ulterior AT one.lt 8 years ago

This seems the correct way to encode ftp url which you could provide for your users:

```
function ftp_url_encode($string) {

  $hex="";
  $retstr = "";
  for ($i=0; $i < strlen($string) ;$i++) {

    $char = $string[$i];
    if(($char >= '0' && $char <= '9') || ($char >= 'A' && $char <= 'Z') || ($char >= 'a' && $char <= 'z') || $char == '.' || $char == '-' ||
$char == '/' || (ord($char) >=128) ) $retstr .= $char;
    else
    $retstr .= "%".strtoupper(dechex(ord($string[$i])));

  }
  return $retstr;
}
```

Browsers mangle certain language characters

▲ -2 ▼ Lucas Gonze -- lucas at gonze dot com 9 years ago

I have attempted to incorporate all of the previous comments, plus several bug fixes, into dphantom's linkencode. I see no bugs for these test cases:
http://example.com/path1;var1=val1/p2;v2
http://example.com/p1;v1/p2;v2
http://[ip:v6:440]:8080
http://example.com:8080
http://example.com/~joe
http://example.com/foobar/~joe
http://username:password@hostname/path 1//path 2/?arg 1=value 1&arg 2=value 2#fragment identifier
hostname/path 1//path 2/?arg 1=value 1&arg 2=value 2#fragment identifier
http://invalid_host..name/

```
function linkencode($p_url){
  $uparts = @parse_url($p_url);

  $scheme = array_key_exists('scheme',$uparts) ? $uparts['scheme'] : "";
  $pass = array_key_exists('pass',$uparts) ? $uparts['pass']  : "";
  $user = array_key_exists('user',$uparts) ? $uparts['user']  : "";
  $port = array_key_exists('port',$uparts) ? $uparts['port']  : "";
  $host = array_key_exists('host',$uparts) ? $uparts['host']  : "";
  $path = array_key_exists('path',$uparts) ? $uparts['path']  : "";
```

```
  $query = array_key_exists('query',$uparts) ? $uparts['query']  : "";
  $fragment = array_key_exists('fragment',$uparts) ? $uparts['fragment']  : "";

  if(!empty($scheme))
    $scheme .= '://';

  if(!empty($pass) && !empty($user)) {
    $user = rawurlencode($user).':';
    $pass = rawurlencode($pass).'@';
  } elseif(!empty($user))
    $user .= '@';

  if(!empty($port) && !empty($host))
      $host = ".$host.':';
  elseif(!empty($host))
      $host=$host;

  if(!empty($path)){
    $arr = preg_split("/([\/;=])/", $path, -1, PREG_SPLIT_DELIM_CAPTURE); // needs php > 4.0.5.
    $path = "";
    foreach($arr as $var){
      switch($var){
      case "/":
      case ";":
      case "=":
        $path .= $var;
        break;
      default:
        $path .= rawurlencode($var);
      }
    }
    // legacy patch for servers that need a literal /~username
    $path = str_replace("/%7E","/~",$path);
  }

  if(!empty($query)){
    $arr = preg_split("/([&=])/", $query, -1, PREG_SPLIT_DELIM_CAPTURE); // needs php > 4.0.5.
    $query = "?";
    foreach($arr as $var){
      if( "&" == $var || "=" == $var )
        $query .= $var;
      else
        $query .= urlencode($var);
    }
  }

  if(!empty($fragment))
    $fragment = '#'.urlencode($fragment);

  return implode('', array($scheme, $user, $pass, $host, $port, $path, $query, $fragment));
}
```

▲ -2 ▼ Andrew Eigus 11 years ago

note that if you implement your own server request engine in the HTTP manner like:

GET $request_uri

you should first split all parts of the $request_uri path and rawurlencode() each part, then concatenate those parts back again. this function will translate the URI correctly:

```
function translate_uri($uri) {
   $parts = explode('/', $uri);
   for ($i = 0; $i < count($parts); $i++) {
     $parts[$i] = rawurlencode($parts[$i]);
   }
   return implode('/', $parts);
}
```

because if you do rawurlencode() over the whole URI, path separator characters '/' are also encoded and request will not happen to be correct. '/' characters should not be encoded, only those parts in between.

hope this helps someone like me...

▲ -2 ▼ ToKaM 3 years ago

Be careful here. rawurlencode changes ä to %C3%83%C2%A4 but firefox changes this internally to %c3%83%c2%a4. This could lead to bugs with rewrite loops.
Cheers.

add a note

Fonctions URL

- base64_decode
- base64_encode
- get_headers
- get_meta_tags
- http_build_query
- parse_url
- rawurldecode
- » **rawurlencode**
- urldecode
- urlencode

Copyright © 2001-2014 The PHP Group My PHP.net Contact Other PHP.net sites Mirror sites

Privacy policy